SIAM J. COMPUT.
Vol. 17, No. 3, June 1988

A POLYNOMIAL APPROXIMATION SCHEME FOR SCHEDULING ON UNIFORM PROCESSORS: USING THE DUAL APPROXIMATION APPROACH*

DORIT S. HOCHBAUM† AND DAVID B. SHMOYS‡

Abstract. We present a polynomial approximation scheme for the minimum makespan problem on uniform parallel processors. More specifically, the problem is to find a schedule for a set of independent jobs on a collection of machines of different speeds so that the last job to finish is completed as quickly as possible. We give a family of polynomial-time algorithms $\{A_\varepsilon\}$ such that A_ε delivers a solution that is within a relative error ε of the optimum. This is a dramatic improvement over previously known algorithms; the best performance guarantee previously proved for a polynomial-time algorithm ensured a relative error no more than 40 percent. The technique employed is the dual approximation approach, where infeasible but superoptimal solutions for a related (dual) problem are converted to the desired feasible but possibly suboptimal solution.

Key words. scheduling, approximation algorithms

1. Introduction. We will consider a fundamental problem of scheduling theory. Suppose that we have a set of jobs J with independent processing times $p_1, \cdots, p_n$ that are to be executed on m nonidentical machines; these machines run at different speeds $s_1, \cdots, s_m$. More precisely, if job j is executed on machine i it takes p_j/s_i time units to be completed. The objective is to assign the jobs to machines so as to minimize the total execution time required to run the jobs assigned to the most heavily loaded machine. In other words, it is the minimum time needed to complete the processing of all of the jobs. In the classification scheme of [GLLR] this problem is denoted $Q\|C_{\max}$, the *minimum makespan problem on uniform parallel machines*. In this paper, we will present a family of algorithms for this problem where these algorithms are, in a sense to be indicated below, the best possible algorithms for this problem.

As is true for most scheduling problems, this problem is likely to be intractable since it is *NP*-complete, and therefore the existence of a polynomial-time algorithm for it would imply that $P = NP$. As a result, the algorithm designer must be willing to settle for a solution somewhat worse than the optimal schedule. One natural approach is to consider *approximation algorithms* for the problem which deliver a schedule with makespan that is guaranteed to be within a specified relative error of the optimum schedule length. This approach was first considered by Graham [G], who showed that if all of the machines have the same speed, then the simple "on-line" procedure of scheduling *any* job when a machine becomes idle always delivers a schedule that finishes within at most $1+(1-1/m)$ times the optimal schedule length. We shall call such a polynomial-time procedure a $(1-1/m)$-approximation algorithm. Work on this special case of identical machines culminated in the recent work of Hochbaum and Shmoys [HS], that showed that for *any* fixed $\varepsilon > 0$ there exists an ε-approximation algorithm. This was a dramatic difference from previously known work on the more general problem with different processing speeds. Although much work had been done

* Received by the editors October 20, 1986; accepted for publication May 13, 1987.

† School of Business Administration, University of California, Berkeley, California 94720. The work of this author was supported in part by the National Science Foundation under grant ECS-85-01988.

‡ Department of Mathematics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139. The work of this author was supported in part by the National Science Foundation under grants ECS-85-01988 and MCS-81-20790, by a Presidential Young Investigator Award, and by Air Force contract AFOSR-86-0078.

on this harder problem (e.g., [CS], [GIS], [HoS]) the best algorithm published to date delivers a solution that could be up to 40 percent more than the optimum [FL].

In this paper, we present a family of polynomial-time algorithms $\{A_\varepsilon\}$, such that the algorithm A_ε is an ε-approximation algorithm for the more general problem. Such a family of algorithms is traditionally called a *polynomial approximation scheme.* Notice that the algorithm A_ε is polynomial in the size of the input, but not in the value of $1/\varepsilon$. If the family of algorithms has the property that A_ε is polynomial in $1/\varepsilon$ and the size of the instance, then the family is known as a *fully polynomial approximation scheme.* Since this problem is strongly *NP*-complete, our results are, however, the best possible in the sense that if there were a fully polynomial approximation scheme for this problem, then $P = NP$ [GJ].

Due to the exponential dependence on $1/\varepsilon$ in the running time of our algorithm, it is not particularly practical for small values of ε. However, the result shows that there do exist polynomial-time algorithms that produce solutions with far superior guarantees to the previously known algorithms, and thus one might hope for practical algorithms with better guarantees than are known today. Note that the discussion above implies that there are limits to the amount of improvement that is possible; still one might hope for an $O(n \log n)$ or $O(n^2)$ algorithm that is guaranteed to have relative error no more than, for example, 5 percent. In addition to this existential sort of practical implication, we also believe that the framework around which our algorithm is built can lead to efficient algorithms with extremely good guarantees. As an example of this, we give an extremely efficient but exceedingly naive algorithm which is an adaptation of our framework, and is guaranteed to deliver a solution that is within 50 percent of the optimum. In addition, the analysis of this algorithm is similarly transparent.

2. A framework for approximation algorithms for scheduling problems. In this section we will describe the basic structure of our polynomial approximation scheme for the minimum makespan problem with uniform processors. Consider for the moment the related question of deciding whether there exists a schedule for a given instance of this problem where all of the jobs are completed by time T. If we think of the units of the processing times p_j as steps, and the units of the speeds as steps per unit of time, then machine i can process Ts_i steps before the deadline T. The decision problem can then be viewed as a *bin-packing problem with variable bin sizes.* Furthermore, notice that the notation for this problem can be simplified by rescaling both the processing requirements and the speeds by a factor of $1/T$: in this bin-packing variant the aim is to decide whether a set of jobs (which we will interchangeably call pieces) of sizes $p_1, \cdots, p_n$ can be packed into a set of bins of sizes $s_1, \cdots, s_m$.

Suppose that we had an efficient procedure for solving the bin-packing problem with variable bin sizes. Then it is possible to solve the minimum makespan problem with uniform machines by a simple binary search procedure. For the midpoint T of the current range of possible optimum makespan values, we run the decision procedure; if a schedule is found then the upper bound is updated to the midpoint, otherwise the lower bound is updated. To initialize the binary search we need to obtain easy upper and lower bounds on the length of the optimum schedule. One such upper bound is $U = \sum_j p_j / \max_i s_i$ (schedule all jobs on the fastest machine) and one such lower bound is $L = \sum_j p_j / (m \cdot \max_i s_i)$ (even if all machines are as fast as the fastest, there must be enough total processing capacity for the machines to process the total processing requirement). Notice that these bounds have a ratio bounded by m. Thus after $\log m + l$ iterations of a binary search procedure the difference between the upper bound and lower bound on the optimum makespan value is at most 2^{-l} times the optimum value.

Unfortunately, this bin-packing problem is also *NP*-complete, so it seems unlikely that we will find an efficient procedure to solve it. Instead we will argue that solving a *relaxed* version of the bin-packing problem will be sufficient for our purposes. We first introduce some useful terminology. For each collection of jobs J with processing times $\{p_1, \cdots, p_n\}$, and set of bin sizes $S=\{s_1, \cdots, s_m\}$, a *truly feasible packing* is a partition of the job set into m parts, B_i, $i=1, \cdots, m$ where the total processing requirement of jobs in B_i is at most s_i for $i=1, \cdots, m$. Similarly, we define an *ε-relaxed feasible packing* to be a similar partition, but one that need only satisfy the weaker (or relaxed) condition that the total processing requirement of jobs in B_i be at most $(1+\varepsilon)s_i$.

An *ε-relaxed decision procedure* is a procedure which, given a collection J of jobs with processing times $\{p_1, \cdots, p_n\}$, and a set of bin sizes $S=\{s_1, \cdots, s_m\}$, outputs one of two outcomes:

(1) An ε-relaxed feasible packing; or

(2) Some certificate that no truly feasible packing exists.

Consider now the binary search procedure described above with an ε-relaxed decision procedure in place of the algorithm assumed to solve the bin-packing problem with variable sizes. Notice that when an update of the lower bound is done, the new value must still be a valid lower bound on the optimum makespan length, since the ε-relaxed decision procedure fails to produce a packing only when no truly feasible packing exists. Similarly, if the upper bound is updated to t, then a schedule has been obtained of length at most $(1+\varepsilon)t$. From these observations it is not hard to obtain the following result.

THEOREM 1. *An ε-approximation algorithm for the minimum makespan problem with uniform parallel machines can be obtained by executing the binary search procedure using an $\varepsilon/2$-relaxed decision initialized with upper and lower bounds, U and L, respectively, for* $\log m + \log(3/\varepsilon)$ *iterations.*

For the complete details of the proof of this, plus a more general setting in which the same basic ideas are applicable, the reader is referred to [HS]. It is also useful to note that by continuing for more iterations (but only polynomially many) it is possible to convert an ε-relaxed decision procedure into an ε-approximation algorithm (as opposed to the result given above which uses an $\varepsilon/2$-relaxed decision procedure). From Theorem 1, we get the following immediate corollary.

COROLLARY 1. *If for all fixed $\varepsilon > 0$ there exists a polynomial-time ε-relaxed decision procedure for the bin-packing problem with variable bin sizes, then there is a polynomial approximation scheme for the minimum makespan problem with uniform parallel machines.*

3. An ε-relaxed decision procedure for bin packing with variable bin sizes.

In this section we will show how to construct an ε-relaxed decision procedure for the bin-packing problem with variable bin sizes for any $\varepsilon > 0$. For the remainder of the paper we will assume that the pieces (or jobs) have sizes $p_1, \cdots, p_n$, and the bins have sizes $s_1, \cdots, s_m$, where $s_1 \geqq s_2 \geqq \cdots \geqq s_m$.

For convenience we shall assume that $1/\varepsilon$ is a positive integer. The description of the algorithm and the proof of its correctness will proceed in a few phases. We first construct a certain layered directed graph with two nodes designated "initial" and "success." We prove that if there is a *truly feasible packing*, then there is a directed path from "initial" to "success." Furthermore, the existence of such a path provides a means of efficiently constructing an ε-relaxed feasible packing. Hence, the procedure consisting of constructing the graph, identifying if there is a path from "initial" to "success," and then deriving the respective packing is indeed an ε-relaxed decision procedure.

An intuitive outline of the algorithm relies on the analogy to the special case of bin packing m bins of equal size $(=1)$. Such an algorithm is given in [HS], but its presentation, however, does not lend itself to the required generalization. Here we modify the description of the algorithm to clarify the analogous procedure in the variable-size bins case. For the equal-size case, all pieces lie in the interval $(0, 1]$, and the attempt is to ε-relaxed pack them in at most m bins. The first pieces to be packed are of size greater than ε; these *large* pieces will be denoted by $J_{\text{large}} = \{j \mid p_j > \varepsilon\}$. The phase of the algorithm where these pieces are packed is called *large-pack.* Since these pieces are large, fewer than $1/\varepsilon$ of them can fit in one bin. These large pieces are further partitioned according to their size in subintervals of length ε^2 each. All piece sizes in such subintervals are all rounded down to the lower end of the subinterval, which is the nearest multiple of ε^2 no more than the original piece size. After this rounding, the number of large piece sizes is at most $w = (1-\varepsilon)/\varepsilon^2$. Thus, the packing of large pieces in a bin can be uniquely described by an array of the distribution of piece sizes that go into that bin. It is an array with one entry for each subinterval, and an integer value between 0 and $1/\varepsilon$ in each entry. Such an array, or a *configuration,* specifies how many pieces of each subinterval go into each bin. A configuration $(x_1, \cdots, x_w)$ is called *feasible* if each $x_i \geqq 0$ and the total sum of the rounded sizes of pieces in the configuration is at most 1, the size of the bin.

The distribution of the remaining large pieces to be packed, the *state vector,* is described by a similar array, except that each entry may contain a nonnegative integer no more than n. Therefore, the total number of possible state vectors is at most n^w. A state vector $(n_1, n_2, \cdots, n_w)$ is *reachable* from a state vector $(n'_1, n'_2, \cdots, n'_w)$ if there is a feasible configuration $(x_1, \cdots, x_w)$ such that $n_i = n'_i - x_i$ for $i = 1, \cdots, w$. The first step of the procedure is to construct a layered directed graph where the nodes correspond to state vectors in the following way. Let $V_0, \cdots, V_m$ be the nodes in the 0th through mth layers, respectively. For $i = 1, \cdots, m-1$, V_i contains a vertex $(i, \mathbf{n})$ for each possible state vector $\mathbf{n}$. V_0 contains only one vertex, the "initial" node, and is labeled with the state vector corresponding to the initial distribution of rounded piece sizes. Similarly V_m contains only the "success" node, which is labeled with the zero state vector (corresponding to the case that all pieces are packed). From each node $(i, \mathbf{n})$, there is an arc directed towards the node $(i+1, \mathbf{n}')$ if and only if the state vector $\mathbf{n}'$ is reachable from the state vector $\mathbf{n}$. Given any truly feasible packing of the original instance, it is easy to see that the induced packing on the (rounded) large pieces implies that there is a path from the initial node to the success node. We next show that from any path from initial to success we can compute an ε-relaxed feasible packing of the large unrounded pieces. The path clearly specifies a packing of the rounded large pieces. If we now restore the large pieces to their original sizes (arbitrarily selecting them from the appropriate subintervals), this "inflating" process may either result in a packing that is truly feasible, or the pieces may exceed the capacity $(=1)$ of some bins. However, the rounding was done in such a way so that it is easy to bound the amount by which the inflated pieces will exceed the bin capacity. To round each piece it was necessary to subtract at most ε^2 from its actual size, and there are no more than $1/\varepsilon$ pieces per bin. Thus, the total difference between actual and rounded piece sizes in a bin is at most ε, and so the total actual piece size cannot exceed $1+\varepsilon$. In summary, the procedure *large-pack* constructs the layered graph, finds a path from initial to success, which then yields an ε-relaxed feasible packing of the large pieces.

We now must show how to extend this ε-relaxed feasible packing to include the small pieces, in a way that will always succeed if there is a truly feasible packing of the original instance. The existence of a truly feasible packing induces a truly feasible packing of the large pieces, with sufficient total slack in the bins in order to accommodate all the small pieces. Moreover, the *total slack* in a truly feasible packing of the large pieces, $V^t = m - \sum_{J \in J_{\text{large}}} p_j$, can be no more than the total slack remaining in bins packed *under capacity* in an ε-relaxed feasible packing of the large pieces, the *relaxed slack*, $V^{\text{rel}} = \sum_{i=1}^m \max\{0, 1 - \sum_{j \in C_i} p_j\}$ where C_i is the set of pieces assigned by *large-pack* to bin i. Now let $J_{\text{small}} = J - J_{\text{large}}$. Since $V^t \geqq \sum_{j \in J_{\text{small}}} p_j$ and $V^{\text{rel}} \geqq V^t$, it follows that $V^{\text{rel}} \geqq \sum_{j \in J_{\text{small}}} p_j$. If this inequality is not satisfied then there can be no truly feasible packing; otherwise we will be able to *small-pack* the remaining (small) pieces. This is done by assigning one piece at a time to any bin with positive slack (that is, filled with less than its unit capacity), even if this slack is less than the size of the piece. This procedure guarantees that:

(1) So long as there is positive slack, small pieces can be packed;

(2) By packing a small piece of size p_j, the total remaining slack is reduced by at most p_j; and

(3) Bins that become packed over capacity in the small-pack phase are packed with at most $1+\varepsilon$ times their true capacity (since the capacity is exceeded only by adding a small piece (i.e., $<\varepsilon$), to a bin that was previously truly feasibly packed).

Therefore, if there is a truly feasible packing, the *large-pack* and *small-pack* procedures will find an ε-relaxed feasible packing. The complexity of the algorithm is dominated by the *large-pack* procedure, where we construct the layered graph. The graph has at most $2+(m+1)n^{1/\varepsilon^2}$ nodes and each node has at most $(1/\varepsilon)^{1/\varepsilon^2}$ arcs originating at it. The construction of each arc amounts to checking the feasibility of the corresponding packing of a bin. This is done with at most $1/\varepsilon$ additions and one comparison. The complexity of the ε-relaxed procedure is hence $O((m/\varepsilon)(n/\varepsilon)^{1/\varepsilon^2})$, which is a polynomial for any fixed positive ε.

In the generalization of the equal-size bin ε-relaxed procedure to the variable-size case we come across a major obstacle. The size of the subintervals in which the pieces are partitioned depends on the size of the bins in which the pieces are to be packed. Moreover, the definition of large and small pieces depends on the size of the bin in which the pieces are to be packed. Let the largest bin size s_1 be normalized to 1. The piece sizes are hence in the interval $(0, 1]$. We round down piece sizes as follows; for a piece in the interval $(\varepsilon^{k+1}, \varepsilon^k]$ we round its size down to the nearest multiple of ε^{k+2}. Formally, define

$$\bar{p}_j = \lfloor p_j / \varepsilon^{k+2} \rfloor \cdot \varepsilon^{k+2}, \quad \text{where } k = \max\{q \geqq 0 \mid p_j \leqq \varepsilon^q\}.$$

Consider now a bin of size s_i in the interval $(\varepsilon^{k+1}, \varepsilon^k]$. In the previous case, the large pieces for this bin were those with sizes in the interval $(\varepsilon s_i, s_i]$ since $s_i = 1$ for all i. This interval can intersect both of the intervals $(\varepsilon^{k+2}, \varepsilon^{k+1}]$ and $(\varepsilon^{k+1}, \varepsilon^k]$ and as a result, we will slightly modify the notion of large in generalizing the algorithm to the case of variable-size bins. The pieces in the interval $(\varepsilon^{k+1}, \varepsilon^k]$ are *large* for this bin whereas the pieces in the interval $(\varepsilon^{k+2}, \varepsilon^{k+1}]$ are called *medium* pieces. The pieces of size less than or equal to ε^{k+2} are *small* for this bin. Notice that the definition of large, medium, and small pieces did not depend on the precise size of the bin, but only on the interval of the form $(\varepsilon^{l+1}, \varepsilon^l]$ that contained the bin size; for convenience we will often refer to this as interval l. As before, it will always be true that a small piece for a bin is no more than ε times the bin size. The strategy will be to execute the packing of the bins $1, 2, \cdots, m$ in two phases:

(1) Large- and medium-piece packing (*l&m-pack*);

(2) Small-piece packing (*small-pack*).

Note that a piece can be large, medium, or small depending on the size of the bin in which it is packed. It will also be useful to classify bin sizes for a given interval of piece sizes. For pieces in interval k, $(\varepsilon^{k+1}, \varepsilon^k]$, a bin is *large* if it is in interval k, *huge* if it is in interval $k-1$, and *enormous* if it is in interval $k-2$.

We construct a directed, layered graph where each node is labeled with a state vector describing the remaining pieces to be packed as large or medium pieces. Since each bin may generate different classes of large or medium pieces, a straightforward representation of such an array leads to roughly n^n such possible state vectors. Instead, the graph will be grouped into *stages*, where a stage will specify the *l&m-pack* of bins in one interval $(\varepsilon^{k+1}, \varepsilon^k]$. Each layer within a stage corresponds to packing a bin in the corresponding interval. Both the bins within the stage and the stages are arranged in order of decreasing bin size. At the end of the *l&m-pack* of the bins of the stage corresponding to interval k, we will provide for the packing of the remaining unpacked pieces in interval k; these pieces will be packed as small pieces. (Note that these pieces must therefore be packed in bins that are enormous for them.)

The treatment of the bins and pieces in this conglomerate way will make it possible to reduce the amount of information encoded in the state vector. The state vector associated with each node is of the form $(\mathbf{L}; \mathbf{M}; V_1, V_2, V)$, where $\mathbf{L}$ and $\mathbf{M}$ are vectors each describing a distribution of pieces in the subintervals of $(\varepsilon^{k+1}, \varepsilon^k]$ and $(\varepsilon^{k+2}, \varepsilon^{k+1}]$, respectively. The subinterval division is of length ε^{k+2} and ε^{k+3} respectively, so each of these vectors is of length $w = (1-\varepsilon)/\varepsilon^2$ and the value of each entry is at most n. For simplicity of notation, let $l_1, \cdots, l_w$ and $\bar{l}_1, \cdots, \bar{l}_w$ denote the values of the lower endpoints of the subintervals of $(\varepsilon^{k+1}, \varepsilon^k]$ and $(\varepsilon^{k+2}, \varepsilon^{k+1}]$, respectively. (See Fig. 1.)

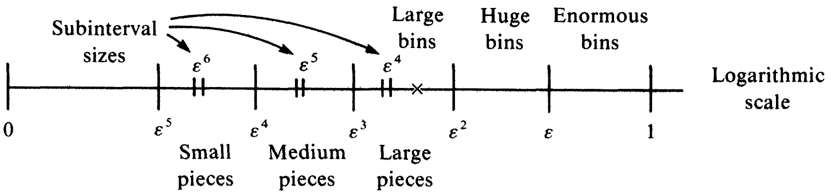

FIG. 1. *The × represents the size of a bin to be packed in interval 2. The identity of pieces and other bins is relative to this interval.*

As was mentioned above, after the *l&m-pack* of the bins in interval k, we must allow for the packing of the pieces in interval k that will be packed as small pieces. These pieces must be packed in enormous bins (those from interval $k-2$) and so we need to know that there is sufficient unused capacity in the enormous bins to at least contain the total size of these unpacked pieces. This is the function of the value V_1 in the state vector; it records the slack, or unused capacity in the partial packing of the enormous bins with large and medium pieces. For stages corresponding to intervals greater than k, we will also need to know the unused capacity in the huge and large bins, and this is the function of V_2 and V, respectively. However, there is a crucial point in that each node is labeled with a possible value for V_1, V_2, and V, so we must be able to represent the possible values in some compact way. Consider the sizes of the pieces that will be packed as small pieces into this as-yet-unused capacity. For V_1, pieces in interval k are small, and thus all pieces to be packed into this unused capacity have rounded sizes that are multiples of ε^{k+2}; as a result, it will be sufficient to represent V_1 as an integer multiple of ε^{k+2}. Similarly, V_2 and V will be represented as integer multiples of ε^{k+3} and ε^{k+4}, respectively. Furthermore, we will argue below (in Lemma 1) that for the purposes of the algorithm, it will be sufficient to have one node for all multiples greater than n/ε^2 for any of these three parameters. As a result, the number of possible state vectors in each layer is bounded from above by $n^{2/\varepsilon^2}\cdot(n/\varepsilon^2)^3$.

Suppose that there are $m_k > 0$ bins with size in $(\varepsilon^{k+1}, \varepsilon^k]$. The stage corresponding to this interval will actually have m_k+1 layers of nodes. In each of these layers, there is one node for every possible state vector $(n_1, \cdots, n_w; \bar{n}_1, \cdots, \bar{n}_w; V_1, V_2, V)$. A node in the $(l+1)$st layer of the stage labeled $(n_1, \cdots, n_w; \bar{n}_1, \cdots, \bar{n}_w; V_1, V_2, V)$ is *reachable* from a node in the lth layer of the stage $(n'_1, \cdots, n'_w; \bar{n}'_1, \cdots, \bar{n}'_w; V_1, V_2, V')$, if:

(1) There is a configuration $(x_1, \cdots, x_w; \bar{x}_1, \cdots, \bar{x}_w)$ where x_i and $\bar{x}_i$ are nonnegative integers for $i=1, \cdots, w$ such that $n_i = n'_i - x_i$ and $\bar{n}_i = \bar{n}'_i - \bar{x}_i$;

(2) The configuration is feasible; that is, the total sum of rounded piece sizes, $\sum_{i=1}^{w} x_i l_i + \sum_{i=1}^{w} \bar{x}_i \bar{l}_i$ may not exceed s, the size of the lth bin in that stage; and

(3) $V = V' + \lceil \text{slack}_l/\varepsilon^{k+4}\rceil$, where $\text{slack}_l = s - (\sum_{i=1}^{w} x_i l_i + \sum_{i=1}^{w} \bar{x}_i \bar{l}_i)$.

We define here the concept of the usable slack in bin l as

$$\text{usable slack}_l = \lceil \text{slack}_l/\varepsilon^{k+4}\rceil \cdot \varepsilon^{k+4}.$$

Note that in the first layer of the first stage we keep only the node with the state vector corresponding to the distribution of pieces in $(\varepsilon, 1]$ and $(\varepsilon^2, \varepsilon]$ with $V_1 = V_2 = V = 0$.

Intuitively, a node z is reachable from an initial state vector for the stage if there is an *l&m-pack* of the first l bins in that group using feasible configurations, leaving a remaining piece distribution as in node z, and accumulating in all l bins a total of $V\cdot\varepsilon^{k+4}$ in usable slack. (See Fig. 2.)

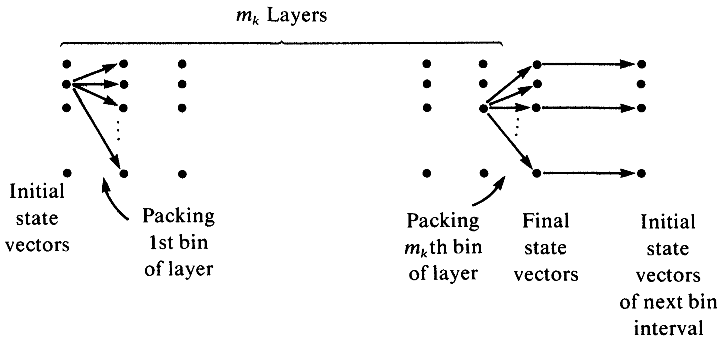

FIG. 2. *One stage of the layered graph (for interval k).*

Note that the number of feasible configurations is at most $(1/\varepsilon^2)^{2/\varepsilon^2}$, since no more than $1/\varepsilon^2$ large or medium pieces can fit in a bin. This number is polynomial for any fixed positive value of ε (or actually just a large constant).

The state vectors in the final layer of one stage have to be updated to the proper form of initial state vectors for the subsequent interval of higher index that contains a bin. Suppose that the next such interval is q intervals away; that is, it is the interval $(\varepsilon^{k+q+1}, \varepsilon^{k+q}]$. Let $l_i^{(t)}$ denote the value of the lower end of the ith subinterval of $(\varepsilon^{k+t+1}, \varepsilon^{k+t}]$ and let $n_i^{(t)}$, $t \geqq 2$, be the number of pieces contained in that subinterval. (For $t = 0, 1$, $n_i^{(t)}$ will be used to denote the large and medium pieces remaining after the *l&m-pack* for interval k.) We will set $q = \infty$ to indicate that there are no more bins.

```
function update $(q, (n_1^{(0)}, \cdots, n_w^{(0)}; n_1^{(1)}, \cdots, n_w^{(1)}; V_1, V_2, V))$
begin
  $V_1^{(1)} := V_1 \cdot \varepsilon^{k+2} - \sum_{i=1}^{w} n_i^{(0)} l_i^{(0)}$
  if $V_1^{(1)} < 0$ then return ("failure")
  if $q \geqq 2$ then
    begin
      $V_1^{(2)} := V_1^{(1)} + V_2 \cdot \varepsilon^{k+3} - \sum_{i=1}^{w} n_i^{(1)} l_i^{(1)}$
      if $V_1^{(2)} < 0$ then return ("failure")
    end
  if $q \geqq 3$ then
    begin
      $V_1^{(q)} := V_1^{(2)} + V \cdot \varepsilon^{k+4} - \sum_{t=2}^{q-1} \sum_{i=1}^{w} n_i^{(t)} l_i^{(t)}$
      if $V_1^{(q)} < 0$ then return ("failure")
    end
  if $q < \infty$ then $V_1 := V_1^{(q)} \cdot \frac{1}{\varepsilon^{k+q+2}}$
  case $(q)$ of
    1: $V_1 := V_1 + V_2;\ V_2 := V$
    2: $V_1 := V_1 + V;\ V_2 := 0$
    $[3, \cdots, \infty)$: $V_1 := V_1;\ V_2 := 0$
    $\infty$: $V_1 := 0;\ V_2 := 0;\ V := 0$
  endcase
  return $((n_1^{(q)}, \cdots, n_w^{(q)}; n_1^{(q+1)}, \cdots, n_w^{(q+1)}; V_1, V_2, 0))$
end
```

The procedure *update* may result in "failure," in which case there is no arc originating at the node examined. Otherwise, the output of the function gives the state vector of the node into which the arc will be directed. It is important to notice that in implementing the procedure we will not scan each subinterval on the line, only those with $n_i^{(t)} > 0$. This is readily done, since the pieces are sorted. Also note that in the updating layer there is at most one arc originating at each node.

Following the updating process of the last interval in which bins are found, we add one more arc level to the final node, "success." A node will have an arc from it to "success" if and only if its state vector is the zero vector $(0, 0, \cdots, 0; 0, \cdots, 0; V_1, V_2, V)$. It is somewhat simpler to avoid altogether the update layer at the end of the last interval and have arcs going directly to "success" based on the outcome of "update" only if the update procedure results in the zero state vector.

LEMMA 1. *The graph constructed has at most* $O(2m \cdot n^{2/\varepsilon^2+3} \cdot 1/\varepsilon^6)$ *nodes.*

Proof. Each of the first and last layers contains exactly one node. All other layers of nodes correspond either to the beginning or the end of a bin interval, or to packing a bin within an interval. For each interval that contains at least one bin, the number of layers is equal to one more than the number of bins in the interval. Since there are m bins the total number of layers is no more than $2m$.

Each layer may contain all possible state vectors. The first $2w \leqq 2/\varepsilon^2$ entries describe the distribution of large and medium pieces remaining in the interval. Every subinterval may contain $r \in \{0, 1, \cdots, n\}$ pieces, so the number of such distributions is $O(n^{2/\varepsilon^2})$.

Finally, V_1, V_2, V express the unused capacities that may be used to pack small pieces. Consider first V_1; since each small piece for these bins is of size at most ε^k, and there are no more than n small pieces, any volume beyond $n\varepsilon^k$ is irrelevant. The value V_1 is actually given as the scalar multiple of ε^{k+2}, so multiples beyond $n\cdot\varepsilon^k/\varepsilon^{k+2} = n/\varepsilon^2$ may be equivalently represented. Identical calculations show that for V_2 and V as well, it is sufficient to consider only n/ε^2 possible values, where the largest value represents all remaining positive volumes. The total number of state vectors is hence $O(2m\cdot n^{2/\varepsilon^2}(n/\varepsilon^2)^3)$, as claimed.

LEMMA 2. *The number of arcs originating at each node is* $O((1/\varepsilon^2)^{2/\varepsilon^2})$.

Proof. Each arc corresponds to a feasible configuration $(x_1, \cdots, x_w; \bar{x}_1, \cdots, \bar{x}_w)$. The number of large pieces from a subinterval x_i that can go in a bin is in $\{0, 1, \cdots, 1/\varepsilon\}$, whereas the number of medium pieces $\bar{x}_i$ can vary from 0 up to $1/\varepsilon^2$. The total number of feasible configurations is therefore bounded by $(1/\varepsilon+1)^{1/\varepsilon^2}(1/\varepsilon^2+1)^{1/\varepsilon}$ which is $O((1/\varepsilon^2)^{2/\varepsilon^2})$.

COROLLARY 2. *The total number of arcs in the graph is* $O(2m(n/\varepsilon^2)^{(2/\varepsilon^2)+3})$.

Once the graph is available, the existence of a path from initial to success can be verified in time linear in the number of arcs. This computation also yields a specific path. We now prove that if there is a truly feasible packing then such a path indeed exists, and if such a path exists, then we can construct a $(2\varepsilon+\varepsilon^2)$-relaxed feasible packing. These two claims are sufficient to yield a $(2\varepsilon+\varepsilon^2)$-relaxed decision procedure.

LEMMA 3. *If there is a truly feasible packing, then there is a path in the multilayered graph from "initial" to "success."*

Proof. Given a truly feasible packing, it is possible to label every piece in the kth interval $(\varepsilon^{k+1}, \varepsilon^k]$ as $L^{(k)}$, $M^{(k)}$, or $S^{(k)}$, depending on the size of the bin in which it is packed:

$j\in L^{(k)}$	if piece j is packed as large—in a bin in interval k;
$j\in M^{(k)}$	if piece j is packed as medium—in a bin in interval $k-1$;
$j\in S^{(k)}$	if piece j is packed as small—in a bin in one of the intervals $0, 1, \cdots, k-2$.

In addition, let $P^{(k)}$ denote all of the pieces in interval k.

Each node in the first layer of a stage is labeled with a state vector that specifies pieces as well as unused capacities. We will show by induction that certain induced partial packings of the truly feasible packing yield paths in the layered graph. The lemma will follow as a corollary to this stronger inductive assertion, which we will give below. Given a truly feasible packing, consider the partial packing induced by the pieces, $M^{(k)}\cup(\cup_{i=0}^{k-1}P^{(i)})$. This leaves a certain amount of slack $v_1^{(k)}$ and $v_2^{(k)}$ in the bins in the intervals 0 through $k-2$ (enormous bins) and in interval $k-1$ (huge bins), respectively.

We claim that there is a path from initial to some node in the first layer of stage k (so that interval k must contain bins) such that the state vector of that node has $V_1 \geqq v_1^{(k)}$, $V_2 \geqq v_2^{(k)}$ and has the piece distribution of $(L^{(k)}\cup S^{(k)}; P^{(k+1)})$.

The claim is certainly true for $k=0$, where there is a trivial path from the initial node to itself, and the initial node does have the appropriate state vector. (Note that $M^{(0)}$ must be the empty set.)

Inductively, we can then assume that at the beginning of stage k (which contains bins) we have a path to a state vector labeled with the as-yet-unpacked pieces from interval k, all of the pieces from interval $k+1$, and appropriate upper bounds on the slacks. Focus now on the packing of the bins in interval k. For each bin we have a truly feasible packing of the large and medium pieces, and this specifies a feasible configuration. This configuration corresponds to an arc in the graph. Now we consider the change in the value of V between the tail and head of this arc. This change is the usable slack (properly represented), which is at least the true slack—for two reasons. First, we effectively round the size of the bin to the next largest multiple of ε^{k+4}, and second we use the rounded (down) sizes of the pieces to compute the usable slack. Thus at each layer within the stage, the value of V in the state vector is at least as large as the actual slack in the truly feasible packing. The reader can verify with little difficulty that following the arcs specified by the feasible configurations generated by the given packing must lead to a node in the final layer of the stage that is labeled with a state vector with piece distribution corresponding to $(S^{(k)};\ L^{(k+1)} \cup S^{(k+1)})$ and where the total usable slack value V is at least the true slack.

It remains only to show that the update arcs are also present as needed. Consider the case where interval $k+1$ also contains bins. In this case, we need only show that the volume represented by V_1 is at least $\sum_{j \in S^{(k)}} \bar{p}_j$, since then there is an update arc, and it leads to a node that satisfies the induction hypothesis. However, this inequality is easy to verify, since $v_1^{(k)} \geqq \sum_{j \in S^{(k)}} p_j$ and we know that $V_1 \geqq v_1^{(k)}$ (by induction) and $p_j \geqq \bar{p}_j$ (by definition). The remaining cases for the size of the next bin (corresponding to $q=2$, $q \in [3, \infty)$ and $q=\infty$) follow by similar calculations.

LEMMA 4. *If there is a path in the graph from initial to success, then there is a* $(2\varepsilon+\varepsilon^2)$*-relaxed feasible packing.*

Proof. There are two types of arcs in the graph, *l&m-pack* arcs that correspond to a feasible configuration, and the update arcs. The relaxed packing is done by tracing the path from initial to success as follows. Each bin has an arc along the path that specifies the *l&m-pack* of pieces in the bin. We pack bins in order of decreasing size (as they are encountered on the path) according to the feasible configuration specified by the arc (arbitrarily choosing pieces from each subinterval). It will be convenient to view the piece as having its rounded size, and later we will consider the effect of inflating it to its true size. In the updating phase at the end of the stage for interval k, we pack the remaining pieces from interval k as small pieces. An unpacked piece j with size in $(\varepsilon^{k+1}, \varepsilon^k]$ is packed in any enormous bin l (in intervals $1, \cdots, k-2$), with *positive* usable slack$_l$. Recall that for a bin interval k its usable slack is computed as if the volume of the bin were rounded to the next highest multiple of ε^{k+4}. We then update usable slack$_l := \max\{\text{usable slack}_l - \bar{p}_j, 0\}$. This *small-pack* phase is always successfully completed since there is an update arc if and only if there is sufficient total slack to accommodate all pieces to be packed as small pieces, and the total usable slack is at least as large. Therefore, we are also able to construct a packing from the path.

Consider the (rounded) pieces packed into bin i where the size of bin i is s_i, which is in interval k. Focus on the small piece j that, when added to the bin, exhausts the usable slack; suppose that piece j is in interval $k+2$. (It certainly cannot be in an interval of smaller index, since it must be small for bin i.) In this case, all pieces in the bin have rounded sizes that are multiples of ε^{k+4}, and in fact, before piece j was added, the rounded piece sizes did not exceed s_i. Hence, the bin contains at most $s_i+\varepsilon^{k+2}$ in terms of rounded sizes. If piece j comes from an interval greater than $k+2$, then the bin clearly contains total rounded size no more than $s_i+\varepsilon^{k+4}+\varepsilon^{k+3}$, which is less than $s_i+\varepsilon^{k+2}$ (since $\varepsilon \leqq 1/2$). Therefore, if B_i is the set of pieces packed in bin i,

$$\sum_{j \in B_i} \bar{p}_j \leqq s_i + \varepsilon^{k+2} \leqq s_i + \varepsilon s_i = (1+\varepsilon)s_i.$$

Furthermore, for any piece j, $p_j \leqq (1+\varepsilon)\bar{p}_j$, since if $p_j \in (\varepsilon^{k+1}, \varepsilon^k]$, $p_j \leqq \bar{p}_j + \varepsilon^{k+2} \leqq \bar{p}_j + \varepsilon\bar{p}_j$. Combining these inequalities, we see that bin j is packed with (unrounded) pieces of total size at most $(1+\varepsilon)(1+\varepsilon)s_i$. This is therefore a $(2\varepsilon+\varepsilon^2)$-relaxed feasible packing.

Thus, the procedure *all-pack* consisting of building the layered graph, finding a path from initial to success, and then (if possible) converting the path to a packing as described above, is a $(2\varepsilon+\varepsilon^2)$-relaxed decision procedure. By observing that $2\varepsilon+\varepsilon^2 \leqq \frac{5}{2}\varepsilon$ when $\varepsilon \leqq \frac{1}{2}$, we see that the following theorem is a corollary of the previous discussion.

THEOREM 2. *For $\varepsilon \leqq \frac{1}{2}$, the procedure all-pack delivers an ε-relaxed packing in $O(m \cdot n^{(10/\varepsilon^2)+3})$ steps. Each step consists of one arc evaluation—at most $(2/\varepsilon^2)-1$ additions and one comparison, or for the update—at most n additions and 3 comparisons.*

4. A $\frac{1}{2}$-relaxed decision procedure for bin packing with variable sizes. In this section we consider a special case of the problem considered in the previous section: we fix $\varepsilon=\frac{1}{2}$. In other words, we wish to construct a procedure which, given an instance $I=(J, S)$, either concludes that no truly feasible packing exists, or else finds a $\frac{1}{2}$-relaxed feasible packing. Unlike the algorithm of the previous section, this algorithm is extremely simple and efficient. Once again, we assume that the bin sizes are $s_1 \geqq s_2 \cdots \geqq s_m$.

Consider the following recursive procedure:

```
procedure pack $(J, S, m)$
begin
  if $\sum_{j\in J} p_j \leqq \sum_{i\in S} s_i$ then
    begin
      $J_{\mathrm{sml}} := \{j \mid p_j \leqq s_m/2\}$
      $J_{\mathrm{new}} := J - J_{\mathrm{sml}}$
      $J_{\mathrm{fit}} = \{j \in J_{\mathrm{new}} \mid p_j \leqq s_m\}$
      if $J_{\mathrm{fit}} \neq$ empty then
        begin
          choose $\bar{j}$ such that $p_{\bar{j}} = \max_{j\in J_{\mathrm{fit}}} p_j$
          pack $\bar{j}$ in bin $m$                (+)
          $J_{\mathrm{new}} := J_{\mathrm{new}} - \{\bar{j}\}$
        end
      $S_{\mathrm{new}} := S - \{m\}$
      if $J_{\mathrm{new}} \neq$ empty then call pack $(J_{\mathrm{new}}, S_{\mathrm{new}}, m-1)$
      while there exists unpacked piece $j \in J$
        find bin $i$ packed with $\leqq s_i$ add piece $j$ to bin $i$   (*)
    end
  else
    output "no truly feasible packing"
end
```

LEMMA 5. *If an instance $I=(J, S)$ has a truly feasible packing then the instance $I_{\mathrm{new}}=(J_{\mathrm{new}}, S_{\mathrm{new}})$ created by procedure pack (J, S, m) has a truly feasible packing.*

Proof. If (J, S) has a truly feasible packing, then certainly so does $(J-J_{\mathrm{sml}}, S)$. Consider any such truly feasible packing. Since all of the pieces in $J-J_{\mathrm{sml}}$ are greater than $s_m/2$ only one piece $\tilde{j}$ can be contained in bin m. If $\tilde{j} \neq \bar{j}$ then form a new packing by interchanging these pieces. By the choice of $\bar{j}$, $p_{\tilde{j}} \leqq p_{\bar{j}}$, so the bin that contained $\bar{j}$ before the interchange remains truly feasibly packed after the swap. Thus, by considering bins 1 to $m-1$, we see that there is a truly feasible packing for the instance I_{new}.

LEMMA 6. *If the procedure pack (J, S, m) outputs "no truly feasible packing" then there is no truly feasible packing.*

Proof. Suppose for a contradiction that there were a truly feasible packing. Then, by Lemma 5, for each recursive call of *pack* there is a truly feasible packing of the specified instance. However, for the failure message to be printed, the last of these instances must have $\sum_j p_j > \sum_i s_i$. This is clearly a contradiction, since no instance that has greater total piece size than total bin size can have a truly feasible packing.

LEMMA 7. *If the procedure pack(J, S, m) does not output "no truly feasible packing" then it successfully packs all pieces in a $\frac{1}{2}$-relaxed feasible packing.*

Proof. In considering the procedure *pack*, there is only one statement in which it could conceivably fail, and this is the statement indicated by $(*)$. Why should it always be possible to find a bin that is packed within its true capacity? If this were not possible, then all bins are packed beyond their true capacity, and then surely $\sum_j p_j > \sum_i s_i$. But this is precisely the situation we have excluded in this case of the **if** statement.

To show that the packing produced is $\frac{1}{2}$-relaxed feasible is also quite simple. Consider the two steps in the procedure in which pieces are packed. In the statement indicated by $(+)$, we ensure that the piece fits within the true bin capacity. In statement $(*)$, we always add to some bin i a piece of size $\leqq s_m/2 \leqq s_i/2$, and since bin i previously contained $\leqq s_i$, afterwards it contains no more than $(3/2)s_i$.

We now discuss an efficient implementation of the procedure *pack*(J, S, m). We shall assume that the piece and bin sizes are given in sorted order. Note that the recursive procedure packs "large" pieces in bins of decreasing bin size, and then packs "small" pieces in bins of increasing size. It will be convenient to maintain two pointers to the sorted list of piece sizes: one to the largest piece no larger than the current bin size, and one to the smallest piece at least half the current bin size. By the monotonicity property just mentioned, only $O(n)$ time is required to maintain these pointers, amortized over the running time of the procedure. Furthermore, given these pointers it is easy to see that the procedure can be implemented in linear time, since no piece need be "touched" more than a constant number of times. By combining these ideas with Lemmas 6 and 7, we get the following result.

THEOREM 3. *The procedure pack(J, S, m) is a $\frac{1}{2}$-relaxed decision procedure for the bin-packing problem with variable bin sizes. Furthermore, given the bin and piece sizes in sorted order, the algorithm runs in linear time.*

5. Conclusions.

In considering the framework employed in the polynomial approximation scheme, it is important to note that this framework is not particular to this scheduling problem. As was discussed in [HS], the key notion in the success of this approach is a *dual approximation algorithm.* For the ordinary bin-packing problem, for example, an ε-dual approximation algorithm delivers a solution where the number of bins used is at most the optimum number, but is possibly infeasible. This infeasibility is bounded: each bin can contain no more than $(1+\varepsilon)$ times the original bin capacity.

The ε-relaxed decision procedure is essentially the same notion, except for the fact that there is no optimization involved. This can be fixed by considering the following generalized problem: given a set of pieces $\{p_1, \cdots, p_n\}$ and a profile of bin sizes $s_1, \cdots, s_m$ find the minimum number of copies of this profile needed to pack all of the pieces. It is quite simple to see how to convert the ε-relaxed decision procedure into an ε-dual approximation algorithm by using binary search.

We believe that this "dual approach" to approximation will continue to yield strong results in constructing approximation algorithms for problems for which good (traditional) approximation algorithms have been, heretofore, elusive. The work presented here certainly adds further confirming evidence.

REFERENCES

[CS] Y. Cho and S. Sahni, *Bounds for list schedules on uniform processors*, this Journal, 9 (1980), pp. 91-103.

[FL] D. K. Friesen and M. A. Langston, *Bounds for multifit scheduling on uniform processors*, this Journal, 12 (1983), pp. 60-70.

[GJ] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.

[GIS] T. Gonzalez, O. H. Ibarra, and S. Sahni, *Bounds for LPT schedules on uniform processors*, this Journal, 6 (1977), pp. 155-166.

[G] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563-1581.

[HS] D. S. Hochbaum and D. B. Shmoys, *Using dual approximation algorithms for scheduling problems: practical and theoretical results*, J. Assoc. Comput. Mach., 34 (1987), pp. 144-162.

[HoS] R. Horowitz and S. Sahni, *Exact and approximate algorithms for scheduling non-identical processors*, J. Assoc. Comput. Mach., 23 (1976), pp. 317-327.

[LL] J. W. S. Liu and C. L. Liu, *Bounds on scheduling algorithms for heterogeneous computing systems*, in Information Processing 74, J. L. Rosenfeld, ed., North-Holland, Amsterdam, 1974, pp. 349-353.

[GLLR] R. J. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Optimization and approximation in deterministic sequencing and scheduling: a survey*, Ann. Discrete Math., 5 (1979), pp. 287-326.